AFRL-IF-RS-TR-1999-147
Final Technical Report
July 1999

UNIVERSITY OF ILLINOIS ATLANTIS SUPPORT

University of Illinois

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. B129

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

DTIC QUALITY INSPECTED 4

19990907 129

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-1999-147 has been reviewed and is approved for publication.

APPROVED: [illegible]

JAMES R. MILLIGAN
Project Engineer

FOR THE DIRECTOR: [illegible]

NORTHRUP FOWLER, III, Technical Advisor
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTD, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNIVERSITY OF ILLINOIS ATLANTIS SUPPORT

Simon Kaplan

Contractor:	University of Illinois
Contract Number:	F30602-94-C-0161
Effective Date of Contract:	15 June 1996
Contract Expiration Date:	14 June 1998
Short Title of Work:	Atlantis - University of Illinois Support
Period of Work Covered:	Jun 96 - Jun 98
Principal Investigator:	Simon Kaplan
Phone:	(217) 244-0392
AFRL Project Engineer:	James Milligan
Phone:	(315) 330-3013

Approved for public release; distribution unlimited. .

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by James Milligan, AFRL/IFTD, 525 Brooks Road, Rome, NY.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE Jul 99	3. REPORT TYPE AND DATES COVERED Final Jun 96 - Jun 98

4. TITLE AND SUBTITLE UNIVERSITY OF ILLINOIS ATLANTIS SUPPORT	5. FUNDING NUMBERS C - F30602-94-C-0161 PE - 62301E PR - B129 TA - 01 WU - 01
6. AUTHOR(S) Simon M. Kaplan	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Illinois at Urbana-Champaign Department of Computer Science 1304 W. Springfield Ave. Urbana, IL 61801-2987	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3801 North Fairfax Drive Arlington, VA 22203-1714 AFRL/IFTD 525 Brooks Road Rome, NY 13441-4505	10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-1999-147
11. SUPPLEMENTARY NOTES AFRL Project Engineer: James Milligan, IFTD, 315-330-3013	
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This project is concerned with developing next-generation collaboration frameworks. Progress in enhancing the usability of collaborative systems hinges on improving our understanding of group work, and applying the resulting insights to the development of collaborate work support frameworks. This work builds on the work of sociologist Anselm Strauss and his notion of "social worlds" where a theoretical framework is applied in developing the Work Locales and Distributed Social Worlds (wOrlds) collaboration system prototype, and the prototype is in turn used to validate and further enhance the theoretical framework. The success of the wOrlds project perhaps can best be measured by the number of complex issues that have been uncovered and the lessons learned in the projects attempt to support the work of distributed social worlds with computer-based solutions.

14. SUBJECT TERMS Computer Supported Cooperative Work (CSCW), Collaboration Environments, Multi-User Domain (MUD)			15. NUMBER OF PAGES 52
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

Abstract

The wOrlds project is concerned with developing next-generation collaboration frameworks. We strongly believe that real progress in enhancing the usability of collaborative systems hinges on improving our understanding of work, and applying the resulting insights to development of collaborative work support frameworks. We are investigating the thesis that appropriate bases for such an approach can be drawn from existing results in sociology, specifically the work of sociologist Anselm Strauss, and his notion of social worlds. In this paper we motivate and overview wOrlds, the collaborative environment we have built in order to explore our ideas and insights. We then critique wOrlds (and, by implication, the class of systems known as MUDs of which it is a member), and point to future directions for investigation.

1 Introduction

For the past several years we have been investigating collaboration frameworks that provide support for the informal, cultural aspects of workaday activities as well as the formal, structured aspects of work traditionally associated with workflow systems and other 'groupware' tools. A major issue in this investigation is the identification of theoretical approaches or models which can inform systems development and function as "bridges" to the ethnographic investiSACgations of work situations currently popular within the Computer-Supported Co-operative Work (CSCW) community. Recently, we have focused on Anselm Strauss' notion of social world (Strauss 1993) as a theoretical model that might address this issue.

Drawing on our prior experience with developing CSCW systems (Kaplan et al 1992), we have constructed a CSCW support environment called wOrlds (Work Locales and Distributed Social Worlds) (Fitzpatrick et al 1995) to evaluate our theoretical approaches and investigate technologies for support of collaborative work.

In its current form, wOrlds has some similarity to a multi-media Multi-User Domain, or MUD (Curtis and Nichols 1994). The multiple locales of wOrlds equate to MUD "rooms" in which groups of people can manipulate shared objects and participate in ongoing audio- and video-conferences. These support the informal aspects of work. wOrlds also can be viewed as a media space (see Dourish 1993, Gaver et al 1995) with sophisticated shared object and navigation facilities due to the pervasive use of audio/video facilities throughout the system. Our initial intention, however, was not to build a MUD (or media space for that matter), but to focus on supporting the activities of social world members through networks of computers.

In this paper, we sketch the origins of the wOrlds project in its predecessor system, ConversationBuilder, and briefly critique its failings and our move towards the notion of social worlds derived from the work of sociologist Anselm Strauss.

We outline our current wOrlds system and the way in which we have interpreted social worlds in the context of computer-based collaboration support. This support is provided by the implementation of locales which support the interactions of social world members. We then critique both the implementation of wOrlds and the concepts used to shape its construction, with a view to identifying some significant open problems in the construction of CSCW technologies and to point the way to possible solutions.

In critiquing our experiences with this system, we challenge the interpretation of the spatial metaphor and propose a move away from space to place and from boundary to centre, where both place and centre are defined from a social world perspective. We also highlight particular problems and challenges in using existing distributed systems infrastructure to support widely distributed collaborative work. The critique of the existing wOrlds system is used to motivate design directions for its successor system, Orbit.

Given the obvious overlap between the current implementation of wOrlds and many MUD-based and media-space collaboration systems, we believe this critique, and the theoretical motivations for which we argue, to be relevant to many systems besides our own.

Finally, we end with a discussion of related work and our conclusions.

2 ConversationBuilder background and motivation

This work grows out of our earlier work on flexible workflow support through a system called ConversationBuilder (Kaplan et al 1992). ConversationBuilder allowed users to describe a wide range of collaborative activities using a protocol specification language loosely based on Speech Acts (Winograd and Flores 1986). The resulting system was used to specify many collaborative activities from asynchronous business processes and software development processes to a complete software development environment.

While ConversationBuilder worked well, it only allowed users to specify the formal aspects of collaborative work. Robinson (1991) has argued that work takes place on two levels simultaneously - a formal level, which emphasizes the manipulation of the artifacts which are an integral part of most work processes, and a cultural level, where the informal aspects of work are played out. These levels are inseparable, each shaping and informing the other. Thus, a primary weakness of the ConversationBuilder system was our failure to take account of the double-level aspects of work.

With the wOrlds project we set out to build a system that affords support for work contexts in all their multifaceted richness, both formal and cultural. One of the major issues we had to face was the identification of appropriate theoretical models of work and workgroups. ('Afford' and 'affordance' are terms drawn from Gibson (1979), and refer to the facilities an environment provides to facilitate the activities of an animal.)

In ConversationBuilder and other workflow systems, (Swenson 1993, Medina-Mora et al 1992, Kogan 1993) actions are the primary building blocks. A workflow system is constructed by specifying the actions that may be performed and (sometimes) the artifacts that are to be manipulated by those actions. Strauss argues convincingly that the actions performed by workers continually vary as workers adapt their activities because of the situated and contingent nature of their work. Thus using actions as a basis for the "codification" of work will be doomed to fail.

Yet, computer programs require that at least some features or assumptions of the system they implement be fixed. We therefore had to identify the key theoretical concepts which we would fix as part of our systems building work. We decided to employ Strauss' concept of social worlds as the basis for our investigations. Detailed motivations and overview of Strauss' theories and concepts can be found in (Fitzpatrick et al 1995).

Strauss' social worlds model provides a rich and multifaceted way of understanding the structure and dynamics of groups. In brief, a social world is defined as a group of individuals (or groups), bonded by a common (and possibly implicit) purpose, and bounded by the limits of effective communication. Members of a social world perform actions to accomplish the shared purpose of the world and these actions will continually permute to suit the contingencies of the situation at hand. Social worlds are not necessarily bounded by traditional social or organizational boundaries: their duration is entirely dependent on the task at hand, and membership in the social world can range from highly informal and/or transient to highly formal and/or persistent.

Strauss' model of social worlds, we should point out, was developed as an abstract way of making sense of his investigations of workplace situations. It was never designed or intended to be used as a kind of "systems architecture" or "systems requirements" framework. Thus our use of Strauss' ideas is part of an amalgam of technical possibilities, past experiences and searches for models that will break through our blindnesses and, in doing so, point the way to potential new solutions to the problem of computer-supported cooperative work.

So, why use social worlds as a basis for building (or, more appropriately, thinking about building) a CSCW system? In large part the answer is historical: Given our previous work in action-based collaboration support, and given the problems that arose in attempting to support only a part of the continuum of the formal aspects of work (ignoring the informal completely), we were searching for a model that displaced action as the central focus of work, admitted the flexibility and contingency of work, and gave the informal aspects of work equal place with the formal. We believe Strauss' approach does all this, and more. For us, Strauss provides a counterpoint to the obsession with action that pervades a significant part of the CSCW community and points the way to an alternative model.

We do not claim, by any means, that we have yet made full use of his ideas and approach. Indeed we would be the first to agree that our use of Strauss thus far has been more as an inspiration than anything else. But we do believe that there is significant value in this approach and that it can act as a lever to break open intellectual logjams. In Section 5 we explain how insights based on a deeper understanding of the subtleties of social worlds are allowing us to move beyond some of the shortcomings of the current version of wOrlds, and identify directions which potentially can help overcome the limitations of wOrlds and similar systems.

3 The wOrlds solution

Our work on wOrlds has proceeded on two parallel tracks that shape and inform each other: a theoretical track developing the notion of locales and a practical track building our prototype system.

3.1 Locales

We have learned from Strauss that as people work, social worlds continually form and dissolve, as needed. For us the essential question for wOrlds was how we could support contexts for work embedded in the computer such that members of social worlds could then use to accomplish their tasks. When groups are working, they need the following:

- The family of artifacts that make up the "formal" layer of their work activities. Examples include program files, medical records, yellow stickies, stripcharts, etc.
- The tools that are used to manipulate these artifacts, such as compilers, editors, debuggers, pens, ECG machines, etc.
- Resources for "effective communication" which grant members of the social world the ability to communicate appropriately to the task at hand.
- Automation of mundane tasks, such as change notifications, where appropriate.
- The ability to navigate, i.e. to seamlessly switch among multiple ongoing tasks, interrelate them as appropriate, and find tasks and people as needed.

The question then becomes: what is it that we can provide through a computer network that allows the construction of contexts of work such that the needs outlined above are met? Our answer is to introduce the notion of locale.

A locale for the purposes of wOrlds is a "virtual space" inside the computer system which is intended to be the vehicle for groups to establish shared contexts. Many locales can exist simultaneously, and context switching among them should be relatively easy. Further, each locale should provide the following:

Furnishings. Access to shared objects and tools which can be manipulated by users as necessary. Although the specific objects and tools used to furnish a locale will vary widely from locale to locale, depending on its purpose, every locale will provide audio and video conferencing among all the users "in" the locale at a given time.

Participants Participants are users who may or may not be present in the locale at any given time, but who have ongoing responsibility, defined by roles, inside the locale.

Visitors. Visitors are users who are in the locale at a given moment, although they may or may not have any particular roles there. Entry to locales can be restricted when required.

Trajectory Schemas or Processes. Processes define the flow of information and control of actions. Processes generally span locales, and can be seen as a way both of automating standard aspects of workaday activities where appropriate and useful, and as a way of grouping locales together,

Actions Actions with user-defined semantics can be invoked at appropriate times, subject to satisfaction of guards and other constraints.

Additionally, locales provide ways of browsing and viewing information, establishing Audio/Video (AV) calls among users and navigating through the collection of locales.

Thus, locales in wOrlds are our current way of "affording" social worlds' interactions via the computer, by providing a framework for construction and support of work contexts.

To support the definition of domain specific locales and the representation of trajectory schemas which exist within and across locales, wOrlds provides a specification language called Introspect (Tolone 1995). We use the term 'trajectory schema' rather than 'process' or 'workflow' to convey the contingent, continually evolving nature of work. Introspect supports run-time modifications to locale definitions and trajectory schema representations. The environment for constructing and modifying such support is itself a locale within wOrlds.

3.2 The Technological basis for wOrlds

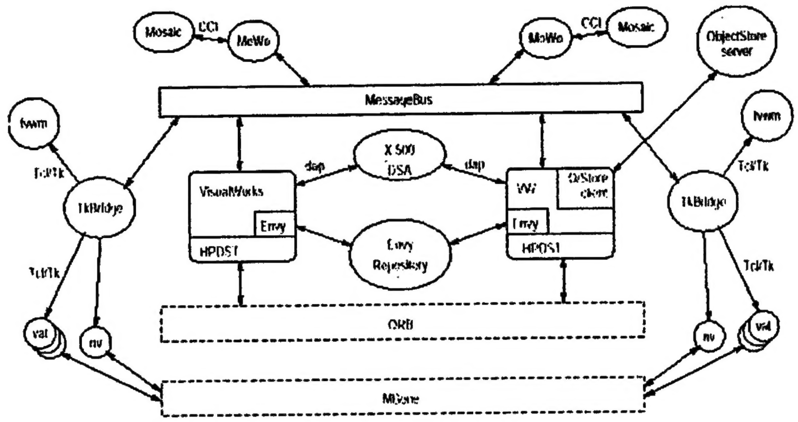
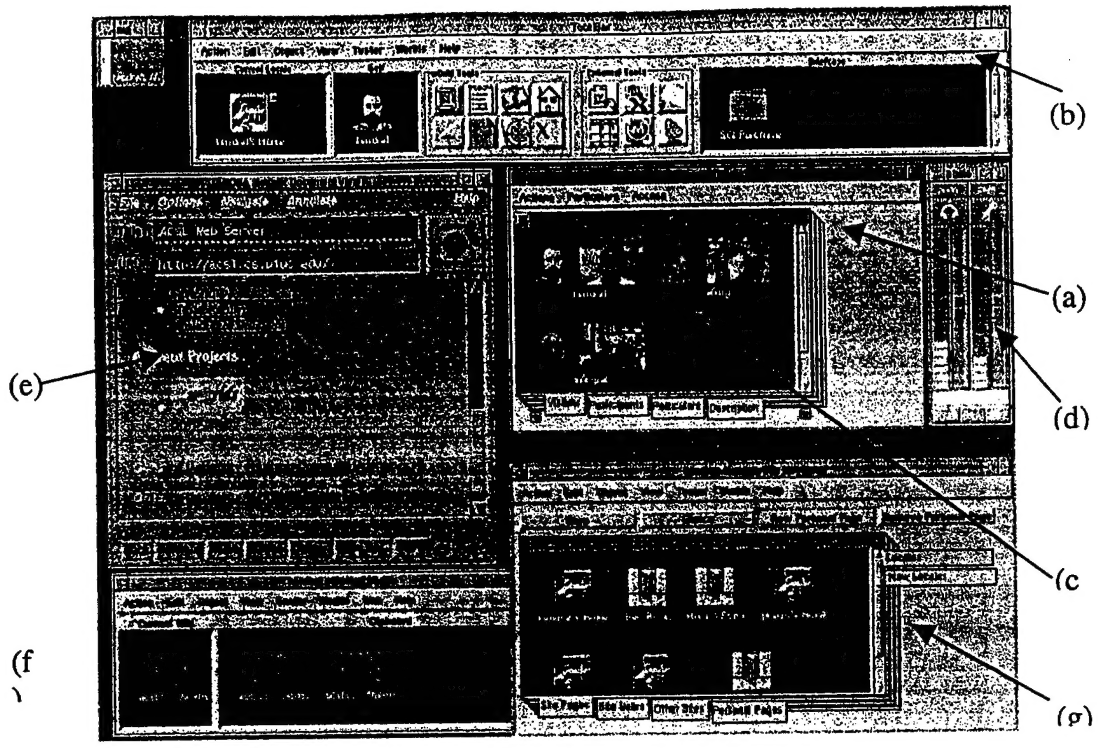


Figure 1 wOrlds Architecture



(a) Locale Pane (c) NV (e) Mosaic (g) Site Navigator
(b) Tool Bar (d) VAT (f) Web Tool

Figure 2. Example Locale Screen Shot

Our goal is to build a system that affords collaboration over the widest possible range of networks and bandwidths, and that scales to support tens of thousands of users spread across thousands of locales. Thus a centralized, server-oriented architecture is unlikely to be appropriate. Instead we've been investigating the use of distributed object frameworks as the basis for building wOrlds. Figure 1 gives a schematic overview of our architecture, based on Object Request Broker (ORB) technology [13].

3.3 A Brief wOrlds Tour

Users enter wOrlds by warping into their home locale. Warping is the most primitive way within wOrlds of moving from one locale to another. It is similar to barging into a room. Users can think of their home locale as their office. An example of a home locale is presented in Figure 2. The display of a locale is minimally characterized by a locale pane and a tool bar.

The locale pane, which is an object shared by all visitors to a locale (see Figure 2(a)), displays to the user:

- video images of those people who are currently present in the locale. Each locale in wOrlds supports an audio/video conference (provided by standard conferencing tools such as NV and VAT, see Figure 2(c) and (d)) to which users are automatically added and removed as they enter and leave the locale, respectively.

- the participants in the locale, whether or not they are present.

- the particulars of the locale. This is the family of shared objects relevant to the locale. We distinguish four types of shared objects:

- administrative objects necessary to the maintenance of the locale (such as the locale pane objects themselves, or role definitions for participants),

- applets, which are small application objects written to furnish the locale (such as an IBIS discussion manager, a shared document annotator or a bug report),

- integrated external tools (such as word processors, calendars, or spreadsheets) and

- external objects (such as files, URLs, or database objects).

- and, a description which provides the purpose or rationale for the locale's existence.

Each of the above is displayed on a separate page of a notebook widget to conserve screen real estate. Pages of the notebook may be 'torn off' whenever a user desires to view multiple parts of the locale pane simultaneously.

The tool bar, see Figure 2(b), has four main components:

- the current locale pane which always contains an iconic representation of the user's current locale. Copies of this icon may be created (e.g., via drag-n-drop) and passed around as references to the locale.

- the self pane which always contains an iconic representation of the user. Copies of this icon may be created and passed around as well.

- a collection of tool icons which provide users with a standard set of tools and actions. Examples include: XEmacs, an issue-based discussion applet, a 'warp to home locale' button, a mailer, a network news reader, a Web Tool which opens a web conference using Mosaic 2.5b3 (see Figure 2(e) and (f)), and a navigation tool.

- a drop area called a briefcase which can accommodate any object you want to carry around from locale to locale.

Unlike a locale pane, the toolbar is unique to a particular user, who can extensively tailor both the bar and the bindings of buttons to tools and applets.

As users work in the wOrlds environment, they can move from locale to locale using the wOrlds' Navigator. An instance of this tool can be created at any time by pressing the appropriate button located on the tool bar. The Navigator, see Figure 2(g) has four components.

- Site Pages: The wOrlds universe is partitioned into many sites. Contained within the site pages are those locales registered at the site the user is navigating. From these pages users can warp or glance other locales. Glancing is a more polite way of entering a locale. For example, users can glance a locale to see who is currently present, at which time, a temporary audio/video connection is established between the glancer and those present in the locale. Those people present in the locale can then warp the glancer into the locale if they so desire. Our glancing model is closely based on the work of (Tang et al 1994).

- Site Users: Contained on this page are icons representing all the users who are registered at the site being navigated. A call feature is provided to allow users to establish locale-independent AV conferences.

- Other Sites: This page contains icons representing other sites in the wOrlds universe.

- Personal Pages: Each user has his/her own set of personal pages which contain a 'hot-list' of locales for that user. The locales, represented by icons on these pages, are not necessarily registered at the site being browsed.

The Navigator is not the only means of navigation within wOrlds. Because wOrlds is MIME-compliant, users and locales can be registered with an HTTP server, accessed from the world-wide web or referenced through mail messages, and treated as URLs.

4 Introspect: The wOrlds Reflective Layer

In order to support the definition of domain specific locales, the representation of the 'process' which exists within and across locales and the modification of each of these at run-time, WORLDS provides a specification language called Introspect. To facilitate the 'radical tailorbility' that support for collaborative work requires, Introspect employs a meta-level architecture, the design of which is based on the principle of reflection. Below we briefly present the design of Introspect's meta-level architecture and demonstrate how this architecture enables run-time modifications within WORLDS.

4.1 Meta-Level Architectures and Reflection

Following Maes' definition of reflection as 'the process of reasoning or acting upon oneself' (Maes 1987), Introspect employs a meta-level architecture where each level, at least in part, is causally connected to its adjacent levels. By definition, two levels are considered causally connected if changes to one level affects the other, and vice versa.

We partition the architecture for Introspect into three separate levels. As Introspect is designed within an object-oriented framework, each level is defined by a collection of objects. Abstractly, we refer to these objects as meta-specifications, specifications and instantiations. We arrange these objects into a meta-object hierarchy such that a meta-specification is a meta-object to a specification and a specification is a meta-object to an instantiation. Furthermore, where each specification is a meta-object to a single instantiation, a meta-specification may be a meta-object to multiple specifications.

The meta-object relationship between meta-specification and specification objects is, on the surface, similar to a prototype relationship., Yet, a specification is not created by directly cloning a meta-specification since this process will produce another meta-specification. More generally, the scope of changes made to specifications and meta-specifications may vary. For example, a locale can be modified by:

- modifying the instantiation of the locale. In this case, such changes are reflected to that locale's specification.

- modifying the specification for the locale. In this case, such changes are reflected to the locale's instantiation and optionally to the locale's meta-specification.

- modifying the meta-specification for the locale. In this case, such changes are optionally reflected to the specifications of that meta-specification and consequently to the instantiations of the specifications.

Thus, as a result of the causal connectivity between the levels of Introspect's architecture, when modifications are made to one level, other levels may be affected. Moreover, as we outline below, users can both specify and modify this support entirely from within the system.

In the following sections we present in more detail three parts of Introspect: the specification of locales, actions and process models. Throughout this presentation we provide examples of how Introspect, using its meta-level architecture, allows run-time modifications to the collaborative support which wOrlds provides.

4.2 Locale Specification

As discussed above, the fundamental component of worlds is a locale. The process of specifying a new locale to worlds begins with the definition of a meta-specification object for the locale. To define a new locale meta-specification users are required to

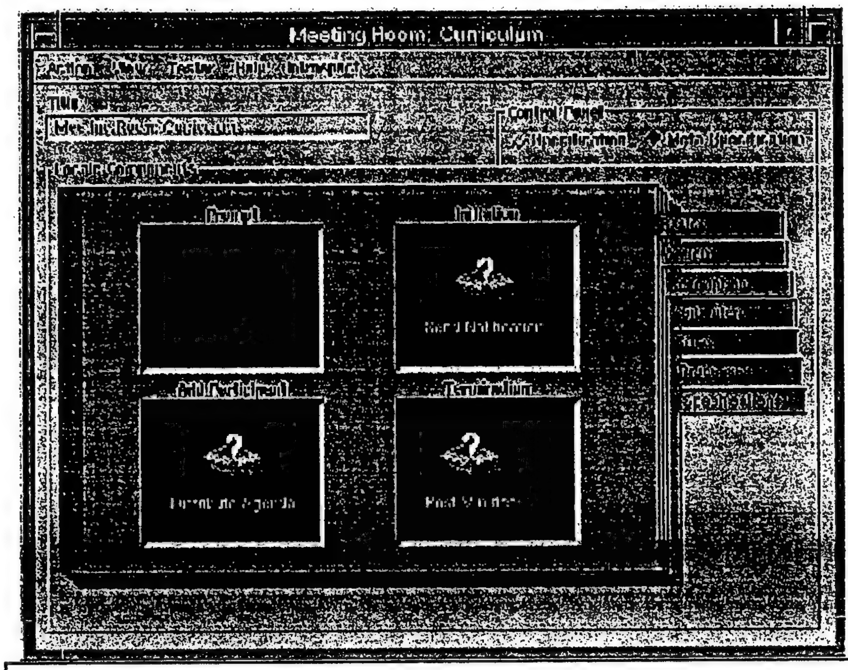


Figure 3. User Interface to a Locale Specification

provide only a title for the meta-specification. Having done so, locale specifications/instantiations may be created from this new meta-specification. Such specifications/instantiations are essentially empty locales which provide only basic domain independent support for collaborative work. Other optional attributes for locale meta-specifications, however, can be defined. These attributes allow domain specific support to be defined. A brief discussion of each attributes is presented below.

- Description: the description outlines the primary work activity for which the locale is created or for which it is being used.
- Particulars: the particulars contains the initial set of particular pages and the objects contained within these pages for any specification/instantiation created from the meta-specification.
- Roles (participants): Locale participants may be partitioned into specific roles. Roles can then be used in the specification of their locale attributes.
- Prompt: Whenever a user creates a new locale specification/instantiation from a meta-specification, wOrlds can prompt the user for additional information.
- Actions: A collection of domain-specific actions can be defined for any locale.
- Initiation, Termination and Add Participant action sequences: An action sequence is essentially a block of code. Whenever a new locale specification/instantiation is

created from a meta-specification, the initiation action sequence is executed in the context of the new locale instantiation. The termination and add participant action sequences for the new locale specification are then initialized to those of its meta-object. The termination action sequence is executed in the context of the locale instantiation just prior to the locale instantiation being destroyed. Whenever a new participant is added to the locale instantiation, the add participant action sequence is executed in the context of the locale instantiation and the new participant.

- Processes: Each locale specification/instantiation, when created, can instantiate a collection of process models.

In Figure 2 we saw a user interface of an example locale instantiation. In Figure 3 we see the interface to a locale specification. This interface consists of a notebook widget and a control panel. Contained within the pages of the notebook widget are the locale attributes described above. For convenience, the control panel allows the users to toggle between viewing the attributes of the locale specification and the attributes of its meta-object, a locale meta-specification. In Figure 3, the user is currently viewing the attributes for the meta-specification.

In Figure 3, we see that the initiation, termination and add participant action sequences have been specified for the locale meta-specification. If a user was to replace, for example, the current termination action sequence with another, she would be prompted to indicate the scope of such changes. The modifications can affect all specifications of the meta-specification, some specifications or only future specifications of the meta-

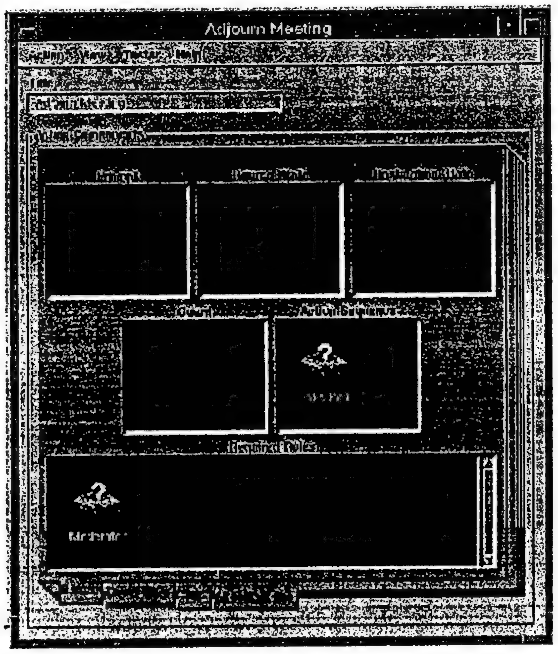


Figure 4 User interface to an action specification

specification.

4.3 Action Specification

Each locale can contain a collection of domain specific actions. In accordance with our meta-level architectural design, users define new domain-specific actions for a locale by creating an action meta-specification. The user interface to an action meta-specification is shown in Figure 4.

As with a locale meta-specification, the only required attribute of an action meta-specification is a title. Other option attributes include:

- Descritiption: Provides the rationale or purpose for the action.
- Help: A textual runtime support for the action to users.
- Prompt. A prompt may be defined to request input from a user upon the execution of an action.
- Roles. Each action can specify a collection of roles in which a user must be playing in order to execute the action.
- Source and Destination States. Any action may be incorporated into a process model. When the source and destination states are specified, the source state must be active in order for the action to be available for execution.
- Action Sequence: An action sequence is essentially a block of code. This action sdequence is executed whenever the action is performed.
- Guard. The guard is an action sequence returning a Boolean value. It must evaluate to true in order for the action to be performed

From action meta-specifications, action specifications and instantiations are constructed. In the same way that the structure of a locale specification is very similar to that of a locale meta-specification, the construction of an action specification is very similar to that of an action meta-specification. the display of an action specification is the same as that shown in Figure 4 except that it contains a control panel for toggling between the specification and meta-specification attributes. the structure of an action instantiation is however much different. This is constructed dynamically and represented simply as a menu item.

This design for locale actions offers us a great deal of flexibility when making modifications. We could, for example, modify the collection of required roles for an action in several different ways:

- by modifying the action specification directly such that the modifications affect only that specification

- by modifying the action specification and propagating the modifications to the meta-specification

- by modifying the meta-specification directly and propagating the modifications to the specifications

- or, by modifying the meta-specification such that only future specifications acquire the modifications

Thus, locale actions, via their attributes, perform several functions. First, they provide information to the users via their description and help. Second, they provide access control via roles, the source state, and guard. Third, they depict temporal relationships within a process model., And fourth, they provide an interface to the action sequence via a prompt.

4.4 Introspect Process Model Specification

Introspect's visual process modeling language is based on Harel's non-overlapping statecharts (Harel 1988). Figure 5 contains the basic building blocks of our process modeling language.

Statechart events, represented as labeled arcs, represent an action within a locale. Single thread process states function as expected according to Harel's statechart definition. Parallel thread process states also function as expected except that each thread in a parallel thread, by definition, contains a distinguished child state called a sync that is used to graphically provide synchronization among process state threads.

Each process state (except syncs) can be attributed with an activation and deactivation state sequence. Also, single thread process states as well as each thread of a parallel

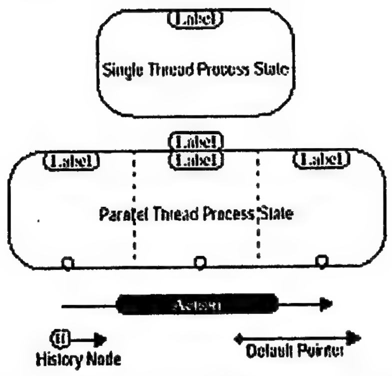


Figure 5 Process Modelling Primitives

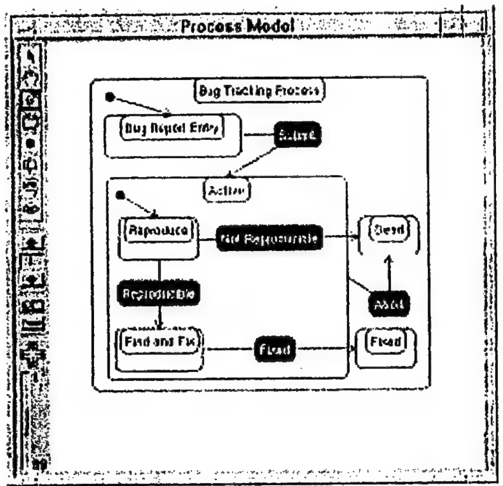


Figure 6 Example Process Model

thread process contains a history node and a default pointer. A history node is a placeholder that records the child state that was last active prior to the parent becoming inactive. Each history node contains an option pointer w indicates the contents of the node prior to the first time a parent state is activated. A default pointer is used to determine the child state to activate when the parent becomes active. When the default pointer points to the history node, this indicates that whenever the parent state becomes active, the child state held in the history node should become active. Figure 6 shows an example process model.

The role of process modeling within wOrlds is threefold. First, process modeling can be used to depict temporal relationships of activities within and among locales. Second, process modeling can be used to bridge locale boundaries to support interactions which arise among locales. For example, Introspect supports actions within one locale that have affects on other locales. Third, process modeling is used to interleave locales to provide users the illusion of participating in multiple locales, simultaneously

5 Obligations: Flexible Workflows with Provable Properties

This section of the paper examines support for more flexible, composable workflows. Computerized support for workflow has changed dramatically in the last fifteen years. Early office procedure and workflow systems, such as SCOOP (Zisman 1977) and Officetalk-D (Ellis and Bernal 1982), required a priori specification of the workflow and modifications were not allowed during execution. While these systems laid excellent groundwork, their inflexibility lead to a lack of acceptance.

More recently, Suchman (1987) proposed a notion of situated actions where actions occur in situ and are essentially ad hoc. Future actions cannot be predicted to every level of detail since it is impossible to know how future situations will influence actions. Even examining an action post hoc requires knowledge of factors that lead up to the action and the situation at the time of the action. As situations change, the people involved normally adjust their actions accordingly. Deciding how to respond to a situation differs depending on the person. For some, deciding on a course of action can be obvious, while for others, how to proceed may be problematic and some deliberation may be required to decide on the next action. When an action is non-problematic, the action simply flows with little concentration on the action itself; whereas, when a problem arises, the details of the action emerge.

It would seem that encoding anything to do with situated actions would be difficult due to the ad hoc nature of actions, the fluidity of a situation, the invisibility of obvious actions, and the fact that activities happen external to a computer and keeping the computer up-todate with the actions taken interferes with the process of acting. Sometimes, however the disadvantages of keeping a computer up-to-date are outweighed by the advantages gained by having a computer assist in tracking activities and acting as a tool for communicating activities among groups of people who are separated by time and/or space.

The notion that actions are situated does not mean that all actions are haphazard, completely unpredictable, or do not follow a plan. Suchman suggests that vague plans of action can be created in advance, realizing both that details of a plan will be filled in as actions progress and that a plan may change over time. Thus, unique situations may alter anything about a plan as needed. In this way, broad plans can be constructed based on past experience or on a proposed method of reaching a goal. Then, as circumstances unfold, exact details can be filled in and the plan can be modified so it continues to provide guidance. A plan does not force a particular set of actions to be taken, but helps provide an orientation to be in the best possible position to take actions when encountering a situation.

Within repetitive circumstances, a vague plan may be shared to guide simultaneous attempts to achieve a similar goal. As the tasks proceed, the vague plan may be changed and, depending on conditions, each individual instance may begin following the changed plan or continue following its current plan.

In this paper, we propose a model which supports both local modifications and general changes to a vague plan. Individual instances can be coded to upgrade to new plans as they become available, remain with their initial plan, or a combination where up until some point in the process a switch will occur, but after that point, the instance maintains the same plan. The implementation of this model, called Obligations, has been prototyped in two collaborative environments, ConversationBuilder (CB) (Kaplan et al 1992) and the new wOrlds environment (Tolone 1995).

The remainder of this paper will proceed as follows. First, our design goals are listed to set the context of the paper. Next, we provide an overview of the terminology used with obligations, define how obligational networks (workflows) are constructed, discuss an error detection mechanism that ensures that executed networks are valid, and describe network execution and manipulation. Finally, the paper ends with a brief description of related works and a conclusion.

5.1 Design Goals

We begin by describing the primary and secondary design goals that pertain to the discussion in this paper.

5.1.1 Primary Design Goals

Support for a Range of Specifications. It is generally accepted that it is impossible to specify, in advance, the full details of how a task will proceed (Suchman 1986, Robinson 1993, Abbott and Sarin 1994). Sometimes participants have a lot of knowledge about what steps are necessary to complete a task, how the steps are related, and what must be accomplished in each step. Other times, very little is known about how to proceed and the flow is essentially designed as the task progresses. Therefore, one necessary goal for a workflow system is to allow initial specifications that are well defined, loosely defined, or incomplete.

Support for Modifications. Regardless of the initial form of a task specification, modifications should be allowed. Conceptually, a task has two types of specifications: general specifications describing how the task should proceed and local specifications resulting from modifications to the general specifications. Modifications can be made to either type of specification. Global modifications potentially change all tasks initiated from a set of general specifications. We say potentially since there are times when grandfather clauses are a valid option and those tasks should continue to follow their initial specifications. As new versions of general specifications become available, users should be allowed to replace a task's current general specifications with new, presumably compatible, specifications. If the two specifications are not completely compatible, local modifications may be necessary to massage the specification into a compatible format without altering any other similar tasks. Local modifications may also be made to align a task with unique circumstances and exceptions.

Inheritance of Existing Definitions. In order to reduce the effort of specifying workflows, a library of specifications should be available. Furthermore, if multiple inheritance is available to tasks and their inherited specifications, complex task definitions can be composed quickly by inheriting existing specifications and then either specializing them or composing them together to make a new, composite specification. To reduce specification effort further, it should be possible to extract a task's specification, either during or after execution, and either save it in a library or use it to instantiate a new task directly. This allows a user to begin with a free-form task, construct a specification as the task progresses, and then reuse the specification for later occurrences of a similar task.

Detection of Errors. Since task specifications are interactively modified by end users, the environment should supply an error detection system that is automatic and as un-intrusive as possible. Error conditions include: tasks that cannot proceed due to lack of specification or dead lock conditions; inheritance of specifications that are incompatible; mismatched signature and call bindings for sub-tasks; and incorrect specification errors. When multiple errors exist in a specification, the error detection system should note as many as possible rather than stopping as soon as one error is detected. In addition, only the affected portions of a task should be halted, allowing work to continue on unaffected portions while errors are fixed.

Bridging of Task Contexts. As described in a previous paper (Bogia et al 1993), we feel it is important to support any ad hoc and/or planned interdependencies that might arise among task contexts. Tasks are not always initiated and completed within the same context. Often it is necessary to place a request for work in another context and then wait for the completion and/or results. Sometimes these dependencies can be templated into a task definition, but often they must be created by users.

5.2 Secondary Design Goals

Many of the secondary design goals described next are mechanisms used to support the primary design goals just mentioned.

Utilization of Meta-objects. We wish to make extensive use of the meta-object concept. A meta-object describes the behavior of an object, and changes to an object's meta-object cause the object to behave differently. When many objects share a meta-object, changes to that meta-object affect all the objects. If temporary changes are desired, a new meta-object can be substituted for the original one for a limited time. The temporary meta-object can call the original meta-object whenever the original behavior is desirable.

Record of History. Given that a user is allowed to modify the workflow while it is running, it is quite possible to expect a person to remove an active unit of the flow and then wish to reinstall it in its pre-deletion state. To support this, some mechanism for tracking state information must be available, independent of the definition of how a task flows (i.e., modifications to the workflow definition do not destroy or lose history information). In addition, since workflows can loop back, any unit in a workflow may be invoked multiple times. The history mechanism should record these invocations and their relationships with other units in the workflow.

Control of Access. Mechanisms for controlling who can make changes and which changes are allowed (e.g., deletion, modification, creation) must exist in the system. Thus, throughout this paper when we talk about people making modifications, it is assumed that they have permission.

5.3 Obligation Overview

This section provides a terminology overview. An obligation represents a request from one person, the obligator, to other agents, the obligatees. Obligations allow users to represent dynamic interdependencies that arise among their activities by specifying the contexts in which the request was initiated and the expected locations where the request can be fulfilled. For a thorough discussion of how obligations interconnect collaborative activities see (Bogia et al 1993). In addition, obligations contain information such as priority, deadline, attachments, a description of the goal to be attained, and the network of sub-obligations that must be performed to complete the obligation. Figure 7 shows a user's view of an obligation in the wOrlds environment.

Figure 8(A) shows a more detailed diagram of the network portion of the obligation shown in Figure 7. The network of an obligation is composed of the following *network objects*
into
smalle
and
inform
ation
cannot
ism
for dist
ions
that sp
s its
access
ther
unless

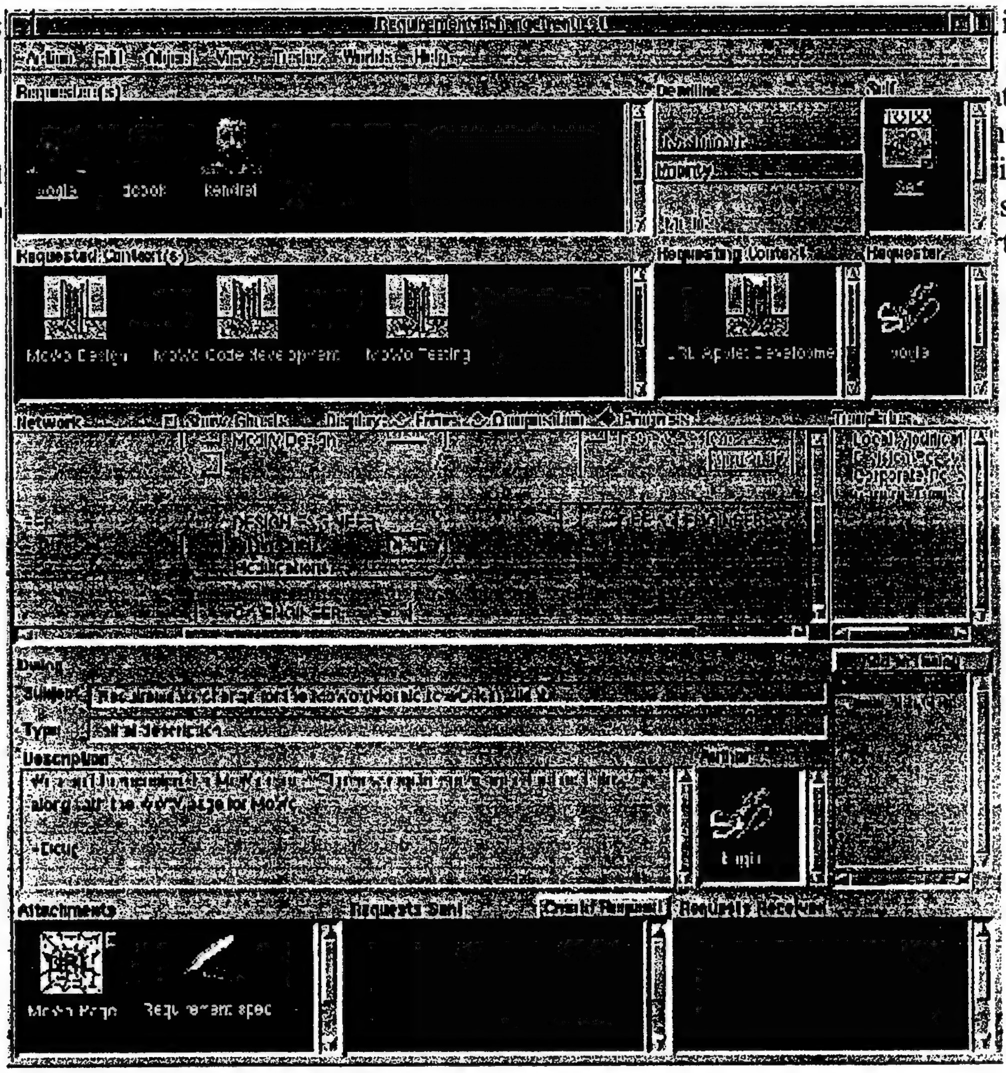


Figure 7 Obligation Screen Dump

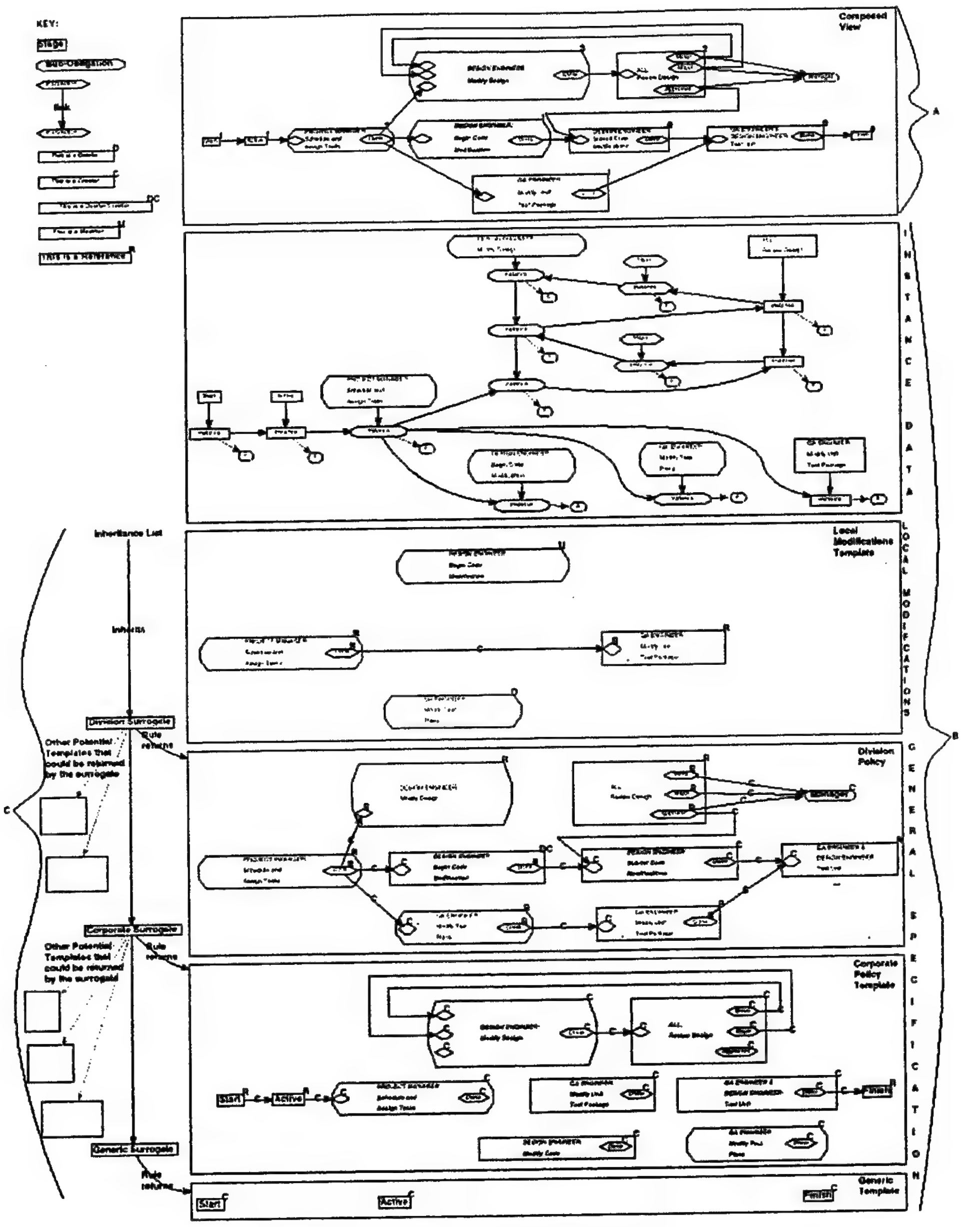


Figure 8. Example Obligation Network

Ports are associated with stages and produce tokens. The concept of an obligation token shares many similarities with petri net tokens. We are exploring how petri nets might be used for static analysis of an obligation. Output ports produce tokens which (potentially) enable attached input ports. Output ports are attached to input ports with links. Multiple links at output and input ports represent forks and joins, respectively. Multiple output or input ports on a stage represent a decision path

Links can also be connected to points outside of the current obligation. Links interconnect two obligations that are already running, send notices to users that obligation events have happened, automatically spawn sub-obligations, and so on.

5.4 The Obligation Network

The following section describes how an obligation's network is constructed from three conceptual layers: an instance data layer, a local modifications layer, and a general specification layer. See Figure 8(B) for a graphical view of the obligation layers. This figure is adapted from the benchmark described in ISPW-6/7 (Kellner et al 1991) which focuses on a late (and isolated) design change in a standard software life cycle.

As a brief overview, the instance data layer maintains an obligation's history. The general specification layer contains a description of the general process that should be followed. The local modifications layer holds any alterations that are specific to an obligation and should not be shared with other similar obligations. Each obligation inherits one or more templates that describe (portions of) the obligation's flow from the local modifications and general specification layers. The order of the multiple inheritance determines how the templates are interspersed.

The following subsections describe each conceptual layer in more detail, introduce how surrogation is used to provide additional flexibility, define the composition algorithm, and then discuss the advantages of this approach

5.5 Obligation Layers

The obligation model separates the templates for a network from the *instance data layer* which consists of information about the current state of the network such as what actions are active, what has been completed, and how many times each network object has been invoked. This layer provides a history record independent of the template layers so that even when a network object is removed, a record of its state prior to removal is maintained.

Each time a network object is activated, a I is created and placed on top of the history stack for the appropriate object. Thus, if a loop back exists within a network, it is possible to have multiple history objects, one for each time through the loop (e.g., in Figure 8(B) two loop backs occurred between Review Design and Modify Design, so there are multiple history objects for each).

The other two layers contain templates that define the general specification and local modifications respectively. The local modifications layer contains one or more templates representing all changes that have been made to the network to bring a specific obligation into alignment with the situation at hand. These changes do not affect any other obligations that have been spawned to accomplish similar goals.

For instance, in this example a decision was made to skip the Modify Test Plans, so it was deleted. A new link was also created connecting directly to the Modify Unit Test Package, so that phase could begin. Of course, as this phase progresses it may be necessary to re-install the Modify Test Plans or to remove the Modify Unit Test Package as well. It is unlikely that any of these modifications should be propagated to all other obligations of this type so they are made within the local layer.

The *general specification layer* contains one or more shared templates that define the intended global process that is to be followed. Any changes made to these templates potentially affect all obligation instances sharing the changed template(s). When a global process is unknown, a free-form template exists which defines a starting and ending point and nothing more.

Within the obligation system, the general specification and local modification templates are treated as metaobjects to an obligation. A different approach would have been to copy all shared templates into a newly instantiated obligation and then all future changes would be local to the obligation. However, we felt that this was unacceptable since the ability to change shared templates and have these changes automatically reflected in running obligations was lost. By separating out the instance data from the templates and using a combination of meta-objects and surrogation (discussed next), obligations can be setup either to act as if templates are copied or to upgrade automatically as shared templates are changed.

5.6 Surrogation

As alluded to earlier, obligation templates do not provide a complete solution to the modification problem since there are also times where the correct policy is to remain with an initial specification (i.e., grandfather clauses). This deficiency is elegantly resolved by using versioning and surrogation. Rather than having a template inherit other templates directly, a template inherits a set of surrogates. A surrogate is an object that acts on behalf of other objects. Typically, surrogation works in conjunction with a version system, using rules to describe which version to select from a version tree. As the version tree changes (e.g., via additions, deletions, modifications to a version's state), the surrogate may return different objects.

For obligations, a surrogate has a rule that locates a template. As conditions change (e.g., new templates are created, the surrogate's rule is changed, or a user changes personal preferences), surrogates may dynamically switch to new templates. However, if dynamic upgrades are not a desirable feature, a surrogate's rule can be set to 'this-one', and the current template is essentially copied into the obligation. The template is only changed if the rule is changed.

This type of 'continual binding' also accomplishes 'delayed binding' where the template to use is selected as late as possible.. A surrogate's rule is first dereferenced at invocation and selects a template according to its rule. The section on Surrogation describes cases where obligations do not upgrade automatically.

However, unlike 'delayed binding', surrogation can continue to select the proper template; thus, the template is automagically maintained during the entire duration of an obligation. For instance, in Figure 8(C) assume that the rule in the divisional surrogate has been defined to return an appropriate policy based on the division(s) of the obligatee(s). If a new version of the divisional policy is released and the surrogate always selects the latest version, then the obligation's network would be upgraded. Another example is to use a surrogate to detect whether a person is a novice or expert and then install a verbose or terse version of a template. As a novice becomes more experienced, the novice setting can be toggled and the network changes to the expert view.

We now describe the algorithm that is used to combine all the templates (via surrogation) into a composite network that can be executed. The algorithm uses a dynamic transparency metaphor where the instance data layer and all templates held in the other two layers are treated as individual transparencies such as the ones used with overhead projectors. Composition is similar to stacking all the transparency sheets, one on top of the other, and then viewing the network. The composition is dynamic in the sense that object alignment across transparencies is done by name unification rather than spatial orientation as would be the case with traditional transparencies.

The topmost template of every obligation is called the master template. This template is always a local modification template and defines the inheritance order of all other templates. The composition algorithm begins by dereferencing all surrogates and generating a precedence ordering of the templates returned. We use a scheme similar to the one used by the Common Lisp Object System (CLOS); however, any other method of converting a multiple inheritance list into a precedence list, such as the methods used by Flavors or LOOPS, could also be used.

The next step is to combine the individual templates into a composite view of the network. As the network is constructed it is cached in the instance data layer by creating a holder object for each valid network object. Each holder object maintains a list of template objects that contribute to its definition. This reduces overhead when attempting to compute the set of methods that should be called when an object is activated, deactivated, receives a token, and so on.

Composition consists of iterating over the list of templates from least specific to most specific (bottom to top in Figure 8). The steps for each layer are:

1. Handle Deleter Objects. A deleter object removes the corresponding holder object if it exist in the obligation under construction. Deleter objects provide a way for more specific templates to remove unnecessary network objects that were created in less specific templates.

2. Handle Creator Objects. A creator object attempts to create a holder object. If the holder object already exists, the creator acts as a modifier object (described later). If the holder does not exist, it is created and the creator template is added to the list of inherited templates for the holder.

3. Handle Deleter/Creator Objects. A deleter/creator object combines the above two operations. This is necessary since within any template only one template object may exist per network object. Deleter/creators allow users to clear prior definitions and redefine the object.

4. Handle Modifier Objects. A modifier object alters portions of an existing holder object. For instance, an example of a modifier is show in the local modifications template of Figure 8(B). This modifier simply changes the existing stage specification into a subobligation without affecting other settings.

5.7 Advantages

There are many advantages to structuring the network using surrogation and multiple layers. First, once a task has been decomposed into at least two layers--which it had to be in order to support a clear distinction between general specifications and local modifications--further decomposition of the layers into multiple templates is relatively trivial.

Multiple templates in the general specifications layer allow participants to use standard multiple inheritance design philosophies when creating their task templates. Furthermore, it is possible to have several similar templates available that can be substituted according to need.

Multiple local modification templates provide a couple of control levels to the obligator[5]. First, each local modification template has associated access controls to tell who may use this template to make changes and what kind of changes (e.g., creation, deletion, or modification) are allowed. Second, by controlling where a local modification template is in the inheritance hierarchy, a person can control the extent of modifications made within a template (i.e., the top-most template can create, delete, or modify anything while the bottom template can only create new objects).

Figure 8 only shows a single local modification, but multiple layers are allowed and they may be interspersed with general specification templates.

A second advantage, as shown in Figure 8, is that while global definitions of general check points can be designed that must be adhered to by all divisions, the general policy does not necessarily dictate the details of how divisions are to move from one point to the next. Each division may then have a template that further refines the network according to the specific way in which work is processed. This refinement continues down to the individual worker.

Third, by allowing multiple inheritance, the effort of specifying complex networks can be reduced either by inheriting a particular template and specializing it or composing a template that specifies phase one with templates that define phase two, three, and so on to create a new work process.

Fourth, since local modifications are separated from the general specifications of a task, it is easy to analyze what changes were made due to localized circumstances. By analyzing the local alterations it is possible to note exceptions that occur frequently and alter the general specification, making it pro-active instead of reactive.
Finally, by maintaining the instance data layer independent of the templates, the extraction of control information from past instances for reuse purposes is easy. This means that any obligation, running or complete, can be used as a template for instantiation or saved into an archive of templates.

5.8 Error Detection

Along with all the flexibility of the obligation system comes the possibility for errors. For instance, the surrogate rules may be wrong, people may make changes to templates that are incompatible with older versions, links may go to non-existent ports, a call to a subobligation may expect one of two results, but receive a third, and the list goes on. Rather than restricting users from making 'invalid' moves, we chose to add an error detection system that warns users that the action(s) just taken possibly put the system in an unacceptable state. Because a history record is held in the instance data layer, it is reasonable to allow modifications that end with an incorrect state since the user can back out of them. The remainder of this section discusses mechanisms for detecting problems in the system. In all cases, if an error is detected, an obligation is placed on the obligator to fix the error. However, having an error does not necessarily stop the obligation from proceeding to completion. We adapted an approach used for visual programming languages such that the paths in error are marked and not allowed to execute; however, the obligation can proceed along other paths that are error free.

5.9 Types of Errors

One type of error detected is when holder objects are left dangling. The composition phase creates holders based on the templates, ignoring the fact that ports must be contained in a stage and that links must connect from a valid output port to a valid input port. Each holder in an obligation is tested to verify that all required objects exist. As an example, consider Figure 8 where the division template is attempting to add a link from the port 'Approved' defined in the corporate template. If the corporate template is changed and the 'Approved' port is removed from the template, then the link defined in the division template will fail to add the link since a link cannot connect to a non-existent port. An error is generated if the link template object has the 'required' property set. If the link is not 'required', then the link is not added and is silently ignored.

Another error that can occur is when a surrogate's rule is meant to return a particular type of template (e.g., a template that defines a division's process for making a design change), but it does not. To detect this, surrogates and templates can specify expected and actual types, respectively. If the two types do not match, an error is generated.

A third type of error detected is when incompatible ports are linked together. It is possible to specify the accepted connections of a port which reflects the type of the connection(s) that can be made similar to the way in which hardware connectors are often shaped differently to keep people from inserting the wrong type of plug. In keeping with our desire to make obligations simple to use, the default type for everything is 'unspecified'; however, judicious use of types when building complex layers of templates allows connection errors to be caught before they become a problem.

There are two types of ports. First, a static port can specify a finite set of accepted connections. When two static ports are linked together their connection sets are intersected and if the result is a non-empty set, the connection is allowed; otherwise, the link is marked as an error. We chose to use finite sets rather than type hierarchies because, as pointed out by Burnett (1993), type hierarchies are generally subsumed by type constraints (finite sets). If multiple links are attached to a port, then each pair of ports must have a non-empty intersection; however, notice that the overall intersection can be empty (i.e., if ports A, B, and C connect to D, then $A \cap D$, $B \cap D$, and $C \cap D$ must be non-empty, but $A \cap B \cap C \cap D$ can be empty). In other words, there must exist a valid connection type for each link, but it does not have to be the same type of connection for all links.

The second option is a polymorphic port. These port connectors attempt to adapt themselves to match all incoming connections; however, unlike static ports, polymorphic ports require that all in-bound types have a common type (i.e., the intersection of all sets is nonempty). Polymorphic connectors specify a local or global variable name, a set of type constraints, and a type constraint function[6]. When a static port is connected to a polymorphic port, the static port's set is first filtered against the type constraints and then intersected with the polymorphic port's set. For instance, assume port A has an accepted connection set of *{x; y; z}*, port B has *{y; z}*, and both connect to the polymorphic port C with a type function of $\cap$, type constraints of *{w; x; z}*, and an initial value of 'unknown'. First, connection A to C is filtered ($\{x; y; z\} \cap \{w; x; z\} = \{x; z\}$) and then intersected with the current value of port C ($\{x; z\} \cap$ 'unknown' is defined to return $\{x; z\}$). Then the B connection is filtered ($\{y; z\} \cap \{w; x; z\} = \{z\}$) and intersected with C's current value ($\{z\} \cap \{x; z\} = \{z\}$) and, thus, C has a connection type of fzg, and both links connecting to C use this type. If a polymorphic port's connection set ever reduces to the empty set, then all links connecting to the port(s) with the given variable are marked as type errors.

If the type variable is a global variable, its scope extends over the entire obligation. Thus, if there are two (or more) global polymorphic ports in an obligation with the same variable name, the type constraints of all the ports must be met and the actual type is defined by the intersection of all connecting types. If the type variable is local, its scope is restricted to the particular port. This is useful for defining ports where the intersection of all connecting types must be non- empty. When two polymorphic ports are connected together, the two variables are unified and all connections must pass all constraints for both variables.

A final type of error is when a sub-obligation or stage completes and a token is not placed on any of the ports that have out-bound links. This situation can arise under several circumstances. One case occurs when the network is being defined as it progresses and out-bound links have not been defined yet. A second case is when a port's criterion is set (possibly erroneously) to allow deadlocks. As an example, consider the 'Review Design' stage again. If the criterion for producing a token is set to majority and three people are requested to review, each of them may decide to respond in a different way. 6Currently the function can be set either to intersect the type constraint set with the connecting types or to subtract the type constraint set from the connecting types.

When this happens, the stage would move to completion since every one has responded, but no tokens would be produced since a clear majority was not reached. Another case is when new options have been added in response to the current circumstances (e.g., a person receives a request for a 'yes' or 'no' vote and adds a new option 'maybe').

Regardless of the reason, an obligation or stage that does not produce a token that (potentially) activates other portions of the network could leave a state where no further progress can be made in the obligation, but the obligatees feel that they have fulfilled the actions requested of them. In these cases, the obligator is notified and requested (obligated) to fix the network to allow it to proceed.

5.10 Example

To increase comprehension, we now describe Figure 8 in detail. The first step in composition is dereferencing each of the surrogates found in Figure 8(C) which returns the templates found in Figure 8(B). These templates are iterated over from bottom to top adding a holder for each creator found in a template, deleting holders for each deleter, recreating a holder for deleter/creators, and modifying information for each modifier. Most of the holder definitions are straight forward since there is a single corresponding creator; however, several special cases exists. First, notice that a deleter/creator is used to rename the 'Modify Code' stage to 'Begin Code Modifications'[7]. Second, the 'Begin Code Modification' stage is further modified to become a subobligation in the local modifications template. Also in that template, the 'Modify Test Plans' sub- obligation is removed, but the links to/from the obligation are not removed by the user, they are detected later the error detection system. The remainder of the template objects are reference objects which hold a place within a template so other objects have a reference point.

The error detection routine detects that two links have been created that do not have valid ports on each end (i.e., the link from 'Schedule...' to 'Modify Test Plans' and from 'Modify Test Plans' to 'Modify Test Unit Package'). These links are quietly removed if 'optional' or cause an error to be generated if either is 'required'. Furthermore, all surrogates are tested to ensure that they returned the proper type of template and all links are tested to see that they connect ports which are connection compatible. \

Once the holders are validated, information from the instance data layer can be used for highlighting particular sub-obligations, stages, ports, or links. For in7There is an internal ID that is consistent between these two objects. stance, in the instance data section, each of the objects that has been instantiated has an associated state (i.e., A - Active or F - Finished). From this information, the objects are rendered using different fonts and the superscripts show the number of times that the object has been activated.

5.11 Network Execution

Once the network is constructed, it can be executed. Network execution consists of tokens being produced and 'mirrored' through the network. By mirroring tokens rather than physically moving them, the tokens remain available in case later network modifications cause new network objects to be attached to an object with a token. For instance, if a new link is attached to an output port that has already produced a token, the link begins mirroring the token immediately. This allows the network to be constructed a piece at a time while continuing to execute as if the links had been there when the output port's token was produced.

Each network object has an associated criterion that specifies when a token is produced. For instance, the default criterion for an input port is to wait until a token is available on all in-bound links. When this occurs, a token is produced which, in turn, activates its associated stage.

Users can also affect network execution by executing actions (e.g., Done, Major, Minor, Approved in Figure 8) on active stages8. When the criterion of an action output stage is met (e.g., all obligatees took the action, one person selected the action, more than half have selected this action), a token is produced.

Tokens in the network are divided into two groups: internal and visible. The internal tokens represent the physical location of all the tokens in the network; however, they are not always visible to the participants of the obligation. For instance, when participants are requested to 'Review Design', it may not be desirable to display the outcome of the vote until the poll closes. In this case, internal tokens move to the appropriate places in the network (e.g., voted: major, minor, or approved), but do not become visible to users or subsequent stages until everyone has made a selection. The internal tokens can be used to test certain conditions (e.g., to test whether everybody has voted) to determine when the visible tokens should be enabled. 8When an obligation is initially created, the Start stage is automatically activated.

5.12 Obligation Manipulation

The flow and location of tokens can be altered by modifying the network. New tokens can be added to the network to enable stages that otherwise might not be enabled. This is beneficial both for debugging purposes (it is not necessary to step through an entire network just to test how latter portions of the net will execute) and for cases when it is necessary, due to time constraints, to start a later stage even though all the dependent stages have not been completed. A new token can be added by inserting an artificial input port history object with a specialized meta-object which always reports that a token is available. Once the network execution actually produces a real token for the input port, its meta-object reverts to the templated version and the port returns to its normal behavior.

Another form of manipulation is altering the network of an obligation. Editing is done by graphical manipulation (e.g., dragging links from one port to another, dropping in new sub-obligations, stages, or ports, or deleting existing objects) on either a running obligation or any of the templates saved in a library. By default, modifications to a running obligation are considered to be local to the obligation and are captured in the topmost master template. However, users may select, via the list to the right of the obligation network (see Figure 7), a particular template to modify. When this is done, the extent of the modification's affect depends on whether the selected template is a local modification template or a general template.

Once an edit is complete, all surrogates that depend on the edited template are notified that they may wish to discard their current template and install the newly modified template. All obligations whose surrogates upgrade to the new template have their networks marked for recomposition.

Notice that if a network modification causes links to be removed, it is possible for a token to be removed from an input port. This happens because links mirror tokens instead of transmitting them. If a link (mirror) is removed, the input port is informed so it can determine how to react. For instance, in the case where the removed link was the sole source of activation for an input port, many possibilities exist. The input port may be set (via properties) to keep a copy of the token and remain active, it could deactivate itself, but upon reactivation return to its state just prior to deactivation, or it could deactivate and when reactivated begin with a brand new history object.

Another form of editing an obligation is 'bottom up' composition where pre-existing obligations are inserted into a network. There are two forms of bottom up composition. The first is to create a one-of link between the two obligations. This allows an ad hoc interdependency to be modelled. The second form adds the sub-obligation's templating information into the parent's network. This means that if the parent's network loops back, new sub-obligations are created automatically. Thus, obligations support top down and bottom up specification. In addition, since the network can be defined by connecting sub-obligations together in any order, obligations also support forward and backward specification.

6 Is wOrlds a MUD?

MUDs, or Multi-User Domains, are environments aimed at supporting groups and oriented around a metaphor of a room. MUDs tend to provide multiple rooms, and facilities to move around from room to room. MUDs were initially developed to support multi-user role-playing games (in these MUDs the rooms might represent different parts of a castle or dungeon). Some MUDs have fixed interfaces and furnishings for rooms, while others allow for varying degrees of programmability, allowing for modification of rooms, interconnections, furnishings and behaviours of the virtual artifacts in the system.

(such MUDs tend to be called MOOs, after the object-oriented style in which they are programmed). Almost all MUDs have a text-based interface, with scrolling descriptions of rooms and keyboard-based actions.

Certainly to readers who have only ever seen text-based MUDs oriented towards game-playing, wOrlds might not seem very MUD-like. However, there are some ways in which wOrlds is MUD-, or MOO-like. Like many MUDs, wOrlds assumes it is useful to divide the universe into collections of essentially independent "rooms", provides facilities for tailoring these rooms in various ways and provides facilities for navigating among these rooms. Like more advanced MUDs, such as PARC's Jupiter (Curtis and Nichols 1994), wOrlds is investigating the integration of media space facilities into rooms, such as audio- and video-conferencing.

The issue of whether wOrlds is or is not a MUD is not interesting in and of itself. The fact that wOrlds ended up being MUD-like is an accident of intersecting research trajectories rather than design. But the practical import of this coincidence is that many aspects of our critique of wOrlds, below, are relevant to other CSCW systems, and the research directions we point towards are potentially the basis for, or part of, a manifesto for an entire class of collaboration systems.

7 Critique and future directions

In this section we will critique wOrlds from the perspectives of both theory and technology. Critiques, almost by definition, tend to focus on the negatives but despite the points and issues raised below we consider the wOrlds experiment to have been successful. We have built a useful CSCW system that has been applied in several different domains. We have been able to demonstrate that a non-action-centered approach to CSCW support can indeed support both formal and cultural aspects of work and we have learned an enormous amount about the subtleties of CSCW support that we simply could never have learned without the hard practical experience of building and working with a system such as wOrlds.

Most importantly, the experience of constructing wOrlds has allowed us to envision its successor system: Orbit, in which we hope to answer many of our own criticisms. Where appropriate we will counterpoint our critique of wOrlds with elements of the planned design of Orbit.

7.1 Positive aspects of wOrlds

By their very nature critiques tend to focus on the negative; here we briefly point out some of the many positive features of the wOrlds system.

Multimedia Rooms wOrlds demonstrated the many benefits of multimedia-based interaction in locales, including the extensive and ubiquitous integration of audio and video.

Shared Objects wOrlds users tend to make extensive use of the underlying shared-distributed-objects-everywhere framework in the system, which supports the ability to create rich tapestries of objects.

Contextual Action. Users found the ways in which wOrlds, especially the Introspect aspects, supported contextual action a useful way of obtaining guidance and having necessary actions at their fingertips.

Accessing People and Locales The ability to call people and glance into or warp to locales is a continually-used feature of the system.

Integrating Mail & Web The seamless integration of email and the web, allowing one to, for example, call someone from a web page or glance a locale from an email message (or vice versa), is a very heavily used feature of the system.

External Object Integration wOrlds users could integrate external objects (files, etc) with their shared ORB-based objects and make extensive use of this facility.

Navigation Facilities wOrlds provided a rich navigation tool for finding users and locales which was much relied on by wOrlds users.

7.2 Theoretical issues

One way of thinking about wOrlds, and MUDs more generally, is that they are about setting boundaries around spaces and then populating those spaces. While we have no problem with populating spaces, we believe that the joint focus on setting boundaries and on spaces are misplaced and lead to fundamental problems with collaboration support. The body of this section really critiques wOrlds indirectly, through discussion of several key concepts which are missing from wOrlds and many other MUD-like systems. In many cases we do not have more than the broadest sense of possible solutions to these issues, they are open problems for us at this time.

7.2.1 Idiosyncrasy and intensity

People are players in multiple social worlds at any point in their lives. The combination of personal history and experience, intersecting social world memberships, and personal desires, needs and goals all combine to mean that people tend to have highly idiosyncratic views of the world. While this is well known, collaboration systems tend to take very little account of this, giving only a fixed group view. Thus, our first criticism of wOrlds is that it does not account for the idiosyncrasies of individuals, allowing each user to build his or her individual view of the system, its locales, and the contents of the locales themselves.

Further, while we are all members of different social worlds the intensity of that membership is again highly idiosyncratic. In one situation one may be intensely involved, and in another have only a passing interest or be interested only in a particular facet of the activities of members of the social world. Our second criticism of wOrlds, thus, that it does not allow for different individuals to participate in locales at differing degrees of intensity: one is either "in" a locale or one is not.

7.2.2 Spaces and places

Like all MUDs, wOrlds provides a collection of "virtual spaces" in which users "gather" to work together, utilizing whatever resources are available in the space. We believe this emphasis on space is a tactical error on the part of CSCW systems designers. While there are many situations in which modeling the physical in the virtual might have advantages, such as modeling meeting rooms, lectures or conferences, there are many times when trying to simulate the physical in the virtual might be positively debilitating.

The telephone is a fine example of this phenomenon. Telephones do not replace face-to-face communication or provide a perfect substitute for it, instead they provide an alternative means of communication, for which we have developed rich social protocols and which have become a critical part of modern society. But thinking of a telephone as a "space" seems ridiculous and there are many other affordances for communication and collaboration that are not spaces, such as mail, fax machines, mailing lists and radio stations and receivers.

We believe that a shift in focus from the purely spatial to a richer notion of place as the basis for supporting collaboration will result in greater power and flexibility in CSCW environments. Places, loosely defined, are collections of affordances with the potential for supporting interaction and communication. Thus, places are a superset of spaces, admitting a wider spectrum of possible bases for collaboration.

One immediate practical upshot of such a shift is that the "fundamental thing" we conceive of collaboration systems as providing becomes much more dynamic. In MUDs, for example, "everything is a room" in the end - the fundamental gathering place for collaboration is the MUD room. In wOrlds, it is the locale, and so forth. While graphical representations of rooms will continue to be useful for many situations, in a place-based system users will be able to use any combination of affordances and representations they see fit for the task at hand. Thus one group may choose to have intermittent AV conferences with a whiteboard while another may choose to use a bulletin board and a third could build an entire "situation room", all to deal with the same problem.

7.2.3 Boundaries, permeability and awareness

People employ a multitude of simultaneous foci in their work. wOrlds and other room-based systems provide rigid rooms with strong boundaries. This makes it very difficult to work on "several things at the same time" or keep one's eye on activities in one situation while concentrating on another.

The essential problem here is that while rooms (and locales) have boundaries, social worlds have centers. Rigid walls around rooms do not permit users to be sufficiently aware of the activities in other spaces. Our challenge, therefore, is to build systems that allow users to have several simultaneous, personalized foci of differing intensity that support the seamless awareness and permeability among work activities which characterize the real, workaday world.

7.2.4 History and trajectory

Finally, a major weakness with almost every collaboration system we know of is that they do not deal well (if at all) with the issue of providing context through history (what has already happened) and trajectory (what might be a good idea to do next).

To a very limited extent workflow systems do provide this in the sense that one can see where one has been (the parts of the workflow that have been completed) and where one might go next (the uncompleted parts of the flow). In (Bogia et al 1995), Doug Bogia outlines a sophisticated and flexible collaboration model, called Obligations, which provides both for histories and for several different kinds of flexible modifications to workflows. However, no support is provided for capturing the cultural-level history and trajectory of work.

We believe that one of the potential advantages of CSCW support is helping users more quickly to acclimatize to new situations. Providing richer notions of history and trajectory is a critical part of this, but at this time we have no ideas beyond Bogia's work as to how to proceed on this. Some possibilities would be to capture the video and audio in a locale, and compress and index this in some way for easy recovery, but the technological problems with storing and indexing vast quantities of (mostly boring) video remain huge.

7.3 Systems implications: abstract view

We do not have, as yet, any clear idea of how to build systems that meet this goal. In our new system, Orbit, we plan to move from the current one-level architecture of wOrlds (objects appear in locales, locales are strictly segmented, users can be in only one locale at a time, and all users see the same view of the locale and its objects) to the more sophisticated three-level architecture sketched in Figure 3. The three levels may be summarized as follows:

7.3.1 Distributed object services

At the lowest level, Orbit should provide a collection of services of various kinds that can be used by workers to accomplish their tasks. Examples include shared, distributed objects, AV conferences, resource discovery agents, tools, etc. This level would also contain the infrastructural services, such as object distribution and replication, which form the technological basis for the system.

7.3.2 Group/locale services.

In Orbit, locales are the places where social worlds can build shared collections of distributed objects that are germane to the task they are trying to perform, available network bandwidth, etc. A locale should be thought of as a shared "lens" which brings certain of the facilities at the lower level together. This level will also contain the services for editing and maintaining locale definitions and processes.

7.3.3 Individual services

Where the distributed services level allows the definition of particular services, objects, etc., and the group/locale level allows the definition of shared group views over subsets of the information at the affordance level, shared processes, etc., the individual services level allows users to define the ways in which they will view and interact with the many locales in which they participate. Unlike wOrlds, which imposed strict boundaries among locales, in Orbit the user will be able to hold information from multiple locales in focus simultaneously, with differing degrees of intensity, and switch easily from one to another.

Naturally the question of how exactly to build a system like Orbit remains open. We will report on our progress in future papers.

7.4 Systems implications: technological view

The issues raised here are those which have presented the wOrlds implementation team with fundamental difficulties or obstacles, as opposed to problems which are, in one way or another, related to particular implementation choices.

7.4.1 Distributed objects

The ability seamlessly to manipulate local and distributed objects is fundamental to the construction of any complex CSCW system. We chose to investigate ORB technology as the basis for the distributed object model in wOrlds. The choice seemed sound - ORBs are built to a standard model, that support the interoperability of systems, languages and machines, and they are the vehicle of choice for many distributed systems researchers and developers. However, we have identified several significant shortcomings with ORB-based distributed object systems which greatly hinder our ability to build a truly scalable collaboration support environment.

7.4.2 Internet instability and ORB reliability.

The Internet provides an extremely unstable networking framework, while ORBs are architected on the assumption that interaction among the various servers providing access to distributed objects is reliable and stable. This creates a fundamental mismatch that makes it extremely difficult to deploy ORB technology over anything other than local area networks (or dedicated wide-area networks with considerable bandwidth). Since the

Internet is not likely to become any more stable in the near future, ORB services that support reliable access to objects is absolutely essential.

7.4.3 Replication

One way that the reliability-of-access issue can be mitigated is to replicate information in many sites. Systems can then concentrate on accessing local data, or have several alternate paths to data, while the underlying replication service takes care of duplicating objects around the network as required. While this sounds very attractive, ORBs support no replication services, and indeed no general replication solutions exist. We believe that the solution to this problem lies not in developing the perfect replication algorithm as every situation could potentially require different replication strategies, but rather to build a replication framework and associated specification environment (rather like Introspect) which allows the development and evolution of many replication strategies.

7.4.4 Adaptability and quality of service

wOrlds should be capable of adapting to changing resource availabilities and different quality-of-service (QoS) demands. wOrlds is built almost exclusively from off-the-shelf components, which provide no facilities for automatically adapting to changing QoS demands, or adapting to changing resources. The entire area of adaptable interfaces which can deal with changes of these kinds remains open, but solutions are critically required in our domain. Obviously any solutions to these problems are going to be closely related to solutions to the problems outlined above..

8 Related work

As we pointed out above, the work most similar to our own is the development of MUDs, such as MediaMOO (Bruckman and Resnick 1993). The major similarity is that MUDs, like wOrlds, reflect the view that the appropriate role of the computer is to provide a setting where users can interact as freely as possible, albeit with text-based interfaces. MUDs also partition the space of interaction. Rooms in MUDs have similar qualities to wOrlds locales: sharing is encapsulated by the boundaries of the room, rooms typically have a purpose and furnishings. Curtis and Nichols' Jupiter adds video-conferencing to the basic MUD providing comparable affordance for informal communication. A key difference is that MUDs are generally implemented on a central server, whereas wOrlds is distributed.

Virtual reality teleconferencing systems such as MASSIVE (Greenhalgh and Benford 1995) and DIVE (Genford and Fahlen 1993) are obviously similar to wOrlds in supporting partitions of the space of interaction and informal, multi-participant, multi-site conferencing. They focus primarily on enabling mixed-media conferencing between multiple participants using heterogeneous interfaces (either text or graphics) and supporting informal intuitions of interaction in a graphically-rendered, non-video system.

These systems provide only extremely limited support for distributed shared objects and integrated shared tools, and make no attempt to model formal work activities.

wOrlds also owes a debt to the development of media spaces specifically at PARC, EuroPARC and Sun, but it extends the basic media space concept with shared objects, locales and many other features. Shared document systems such as GroupDesk (Fuchs et al 1995) and BSCW (Bentley and Dourish 1995) provide useful abstractions for cooperative work based around documents, DIVA (Sohlenkamp and Chwelos 1994) adds a notion of rooms and shared audio-video conferencing.

Although we have focused on Strauss' social worlds model as the basis for our investigations, this work does not exist in isolation. A significant portion of the theoretical work in the CSCW community is, by and large, in fundamental alignment with Strauss' approach, even if the social worlds of the researchers may not overlap. For examples, see (Bowers et al 1995, Robonson 1993, Schmidt 1994, Starr and Ruhleder 1994).

9 Conclusions

The success of the wOrlds project notwithstanding, for us the real value of this project has been the complex of difficult issues which the project has opened up. On the theoretical front it has exposed us to the subtleties of social worlds as a way of understanding group interactions and raised many issues concerning appropriate support for social world interactions through the computer. Building wOrlds has led to the uncovering of a range of issues concerning distributed systems infrastructure. We believe that the successful future of CSCW systems depends in no small part on finding solutions to these problems.

10 Acknowledgments

This work has been supported by the National Science Foundation under grants CDA-9401124, CCR-9108931 and CCR-9007195, the Defense Advanced Research Projects Agency under grant F30603-94-C-0161, the US Army Corps of Engineers Research Laboratory, Fujitsu Open Systems Solutions Inc., Intel, Bull, Sun Microsystems, Hewlett-Packard, Digital Equipment and the Department of Computer Science of the University of Queensland. The work reported in this paper has also been funded in part by a Co-operative Research Centre Program through the Department of the Prime Minister and Cabinet of Australia. Geraldine Fitzpatrick has received additional support through an Australian Postgraduate Research Award and an Australian Telecommunications and Electronics Research Board scholarship.

The views and conclusions expressed in this document are those of the authors and should not be interpreted as representing the official policies, expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

Online information about the wOrlds project, systems availability and references to the technology referred to in this paper can be obtained from: http://www.dstc.edu.au/TU/wOrlds/

11 References

Abbott, K. and Sarin, S. (1994). Experiences with workflow management: Issues for the next generation. Proceedings ACM Conference on Computer Supported Cooperative Work (CSCW), Chapel Hill, NC, pp 113-120, October 1994.

Benford, S. and L. Fahlen (1993). A Spatial Model of Interaction in Large Virtual Environments. Third European Conference on Computer-Supported Cooperative Work (ECSCW 93), Milan, Italy, Kluwer Academic Publishers.

Bentley, R. and P. Dourish (1995). Medium Versus Mechanism: Supporting Collaboration through Customization. Fourth European Conference on Computer-Supported Cooperative Work, Stockholm, Sweden, Kluwer Academic Publishers.

Bogia, D., Tolonw, W. Kaplan, S. and de la Tribouille, E. (1993), Supporting Dynamic Interdependencies among Collaborative Work Activities, Proceedings ACM Conference on Organizational Computing Systems, Milpitas, CA, pp 108-118.

Bogia, D. P. (1995). Obligations: Flexible Support for Dynamic Workflows. Computer Science Department,. Urbana, IL, USA, University of Illinois.

Bowers, J., G. Button, et al. (1995). Workflow from Within and Without: Technology and Cooperative work on the Print Industry Shopfloor. Fourth European Conference on Computer-Supported Cooperative Work (ECSCW '95), Stockholm, Sweden, Kluwer Academic Publishers.

Bruckman, A. and M. Resnick (1993). Virtual Professional Community: Results from the MediaMOO Project. Third International Conference on Cyberspace, Austin, TX.

Burnett, M (1993), Types and Type Inference in a Visual Programming Language, Proceedings IEEE Symposium on Visual Languages, Bergen, Norway, 1993.

Curtis, P. and D. A. Nichols (1994). MUDs Grow Up: Social Virtual Reality in the Real World. 1994 IEEE Computer Conference, IEEE Press.

Dourish, P. (1993). Culture and Control in a Media Space. Third European Conference on Computer-Supported Cooperative Work (ECSCW 93), Milan, Italy, Kluwer Academic Publishers.

Ellis, C. and Bernal, M. (1982). OfficeTalk-D: An experimental Office Information System, Proceedings ACM SIGOA Conference on Office Information Systems, pp 131-140, June 1982.

Fitzpatrick, G., W. J. Tolone, et al. (1995). Work, Locales and Distributed Social Worlds. Fourth European Conference on Computer-Supported Cooperative Work, Stockholm, Sweden, Kluwer Academic Publishers.

Fuchs, L., U. Pankoke-Babatz, et al. (1995). Supporting Cooperative Awareness with Local Event Mechanisms. Fourth European Conference on Computer-Supported Cooperative Work, Stockholm, Sweden, Kluwer Academic Publishers.

Gaver, W. W., G. Smets, et al. (1995). A Virtual Window on Media Space. Conference on Human Factors in Information Systems (CHI '95), Denver, Co, ACM Press.

Gibson, J. J. (1979). The Ecological Approach to Visual Perception. New York, Houghton-Mifflin.

Greenhalgh, C. and S. Benford (1995). Virtual Reality Tele-Conferencing: Implementation and Experience. Fourth European Conference on Computer-Supported Cooperative Work, Stockholm, Sweden, Kluwer Academic Publishers.

Group, O. M. (1995). The Common Object Request Broker: Architecture and Specification Revision 2.0, Object Management Group, Inc.

Harel, D. (1988). On visual formalisms. Communications of the ACM 31(5), pp 514-530, May 1988.

Kaplan, S. M., W. J. Tolone, et al. (1992). Flexible, Active Support for Collaborative Work with ConversationBuilder. ACM Conference on Computer-Supported Cooperative Work (CSCW 92), Toronto, Canada, ACM Press.

Kellner, M., Feiler, P., Finkelstein, A., Katayama T., Osterweil L, Penedo M., and Rombach H., Software Process Example for ISPW 7. Proceedings 7th International Software Process Workshop, IEEE Press, Yoountville, CA, 1991.

Kogan, D. (1993). Design and Implementation of CB Lite. ACM Conference on Organizational Computing Systems (COOCS 93), Milpitas, CA, ACM Press.

Maes, P. (1987). Concepts and Experiments in Computational Reflection. ACM Conference on Object Oriented Programming Languages and Systems (OOPSLA), pp 147-155. October 4-8, 1997.

Medina-Mora, R., T. Winograd, et al. (1992). The Action Workflow Approach to Workflow Management Technology. ACM Conference on Computer-Supported Cooperative Work (CSCW 92), Toronto, Canada, ACM Press.

Robinson, M. (1991). "Double-Level Languages and Cooperative Working." AI and Society 5: 34-60.

Robinson, M. (1993). Design for Unanticipated Use. Third European Conference on Computer-Supported Cooperative Work (ECSCW 93), Milan, Italy, Kluwer Academic Publishers.

Schmidt, K. (1994). The Organization of Cooperative Work: Beyond the 'Leviathan' Conception of the Organization of Cooperative Work. ACM Conference on Computer-Supported Cooperative Work (CSCW '94), Chapel Hill, NC, ACM Press.

Sohlenkamp, M. and G. Chwelos (1994). Integrating Communication, Cooperation and Awareness: The DIVA Office Environment. ACM Conference on Computer-Supported Cooperative Work (CSCW '94), Chapel Hill, NC, ACM Press.

Starr, S. L. and K. Ruhleder (1994). Steps Towards and Ecology of Infrastructure: Complex Problems in Design and Access for Large-Scale Collaborative Systems. ACM Conference on Computer-Supported Cooperative Work (CSCW '94), Chapel Hill, NC, ACM Press.

Strauss, A. (1993). Continual Permutations of Action. New York, Adeline De Gruyter.

Suchman, L. (1987), Plans and Situated Actions, Cambridge University Press.

Suchman, L. (1993). Do Categories Have Politics: The Language/Action Perspective Reconsidered. Third European Conference on Computer-Supported Cooperative Work (ECSCW 93), Milan, Italy, Kluwer Academic Publishers.

Swenson, K. (1993). Visual Support for Reengineering Work Processes. ACM Conference on Organizational Computing Systems (COOCS 93), Milpitas, CA, ACM Press.

Tang, J., E. Isaacs, et al. (1994). Supporting Distributed Groups with a Montage of Lightweight Interactions. ACM Conference on Computer-Supported Cooperative Work (CSCW '94), Chapel Hill, NC, ACM Press.

Tolone, W. J., S. M. Kaplan, et al. (1995). Specifying Dynamic Support for Collaborative Work within wOrlds. ACM Conference on Organizational Computing Systems (COCS '95), Milpitas, CA, ACM Press.

Winograd, T. and F. Flores (1986). Understanding Computers and Cognition. Reading, New York, Addison-Wesley.

Zisman, M. (1977). Representation, Specification and Automation of Office Procedures, PhD Thesis, Wharton School, University of Pennsylvania.

DISTRIBUTION LIST

addresses	number of copies
JAMES MILLIGAN 525 BROOKS RD ROME NY 13441-4505	5
UNIV OF ILLINOIS URBANA-CHAMPAIGN DEPT OF COMPUTER SCIENCE 1304 W. SPRINGFIELD AVE URBANA IL 61801-2987	4
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	2
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
ATTN: NAN PFRIMMER IIT RESEARCH INSTITUTE 201 MILL ST. ROME, NY 13440	1
AFIT ACADEMIC LIBRARY AFIT/LDR, 2950 P.STREET AREA B, BLDG 642 WRIGHT-PATTERSON AFB OH 45433-7765	1
AFRL/MLME 2977 P STREET, STE 6 WRIGHT-PATTERSON AFB OH 45433-7739	1

AFRL/HESC-TDC 1
2698 G STREET, BLDG 190
WRIGHT-PATTERSON AFB OH 45433-7604

ATTN: SMDC IM PL 1
US ARMY SPACE & MISSILE DEF CMD
P.O. BOX 1500
HUNTSVILLE AL 35807-3801

TECHNICAL LIBRARY D0274(PL-TS) 1
SPAWARSYSCEN
53560 HULL ST.
SAN DIEGO CA 92152-5001

COMMANDER, CODE 4TL000D 1
TECHNICAL LIBRARY, NAWC-WD
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6100

CDR, US ARMY AVIATION & MISSILE CMD 2
REDSTONE SCIENTIFIC INFORMATION CTR
ATTN: AMSAM-RD-OB-R, (DOCUMENTS)
REDSTONE ARSENAL AL 35898-5000

REPORT LIBRARY 1
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545

ATTN: D'BORAH HART 1
AVIATION BRANCH SVC 122.10
FOB10A, RM 931
800 INDEPENDENCE AVE, SW
WASHINGTON DC 20591

AFIWC/MSY 1
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016

ATTN: KAROLA M. YOURISON 1
SOFTWARE ENGINEERING INSTITUTE
4500 FIFTH AVENUE
PITTSBURGH PA 15213

USAF/AIR FORCE RESEARCH LABORATORY 1
AFRL/VSOSA(LIBRARY-BLDG 1103)
5 WRIGHT DRIVE
HANSCOM AFB MA 01731-3004

ATTN: EILEEN LADUKE/D460 1
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

OUSD(P)/DTSA/DUTD 1
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.